DEEP LEARNING FOR ECONOMICS AND FINANCE

University of Geneva, Switzerland

February 26th – 28th & March 25th – 27th, 2025

https://github.com/sischei/Deep_Learning_Geneva_2025

Simon Scheidegger
simon.scheidegger@unil.ch

A DENSE PROGRAM

Day 1, Wednesday, February 26th, 2025 (Pavillon Mail PM03)

Time	Main Topics
09:00 - 10:30	Introduction to Machine Learning and Deep Learning (part I)
10:30 - 10:45	Coffee Break
10:45 - 11:45	Introduction to Machine Learning and Deep Learning (part II)
11:45 - 12:15	Python refresher, and basics on the Cloud infrastructure

A DENSE PROGRAM

Day 2, Thursday, February 27th, 2025 (Pavillon Mail PM03)

Time	Main Topics
09:00 - 10:30	A hands-on session on Deep Learning, Tensorflow, and Tensorboard
10:30 - 10:45	Coffee Break
10:45 - 12:15	Introduction to Deep Equilibrium Nets (DEQN)
12:15 - 13:30	Lunch Break
13:30 - 14:15	Hands-on: Solving a dynamic model with DEQNs
14:15 - 15:00	Hands-on: Solving a dynamic stochastic model with DEQNs
15:00 - 15:15	Coffee Break
15:15 - 16:00	Exercise: Solving a dynamic stochastic model by example
16:00 - 16:45	Introduction to a tuned DEQN library: solving a stochastic dynamic OLG model with an analytical solution

A DENSE PROGRAM

Day 3, Friday, February 28th, 2025 (Pavillon Mail PM03)

Time	Main Topics
09:00 - 10:30	Surrogate models part I: (for structural estimation and uncertainty quantification via deep surrogate models), with an example DSGE model solved with DEQN and pseudo-states
10:30 - 10:45	Coffee Break
10_45 - 12:15	Surrogate models part II: (for structural estimation and uncertainty quantification via Gaussian process regression

A DENSE PROGRAM

Session 2

Day 4, Tuesday, March 25th, 2025 (Uni Dufour U364)

Time	Main Topics
09:00 - 10:30	Recap of Session 1 (Neural Networks, Gaussian Processes, DEQN, Surrogate Models)
10:30 - 10:45	Coffee Break
10:45 - 11:30	Bayesian Active Learning and GPs for Surrogate Models
11:30 - 12:00	Creating GP-based surrogates from DSGE models
12:00 - 12:15	Gaussian Process Regression in Finance: From Dynamic Incentive Models to Portfolio Optimization (if time permits)

A DENSE PROGRAM

Day 5, Wednesday, March 26th, 2025 (Uni Dufour, Salle 408)

Time	Main Topics
09:00 - 09:45	Introduction to the macroeconomics of climate change, and integrated assessment models
09:45 - 10:30	Solving dynamic stochastic, nonlinear, nonstationary models, with an application to Integrated Assessment Models
10:30 - 10:45	Coffee Break
12:15 - 13:00	Solving the (non-stationary) DICE model with Deep Equilibrium Nets
12:30 - 13:30	Lunch Break
13:30 - 15:00	Putting things together: Deep Uncertainty Quantification for stochastic integrated assessment models
15:00 - 15:15	Coffee Break
15:15 - 16:45	Solving PDEs with Partial Differential Equations with NNs (PINNs) (I)

A DENSE PROGRAM

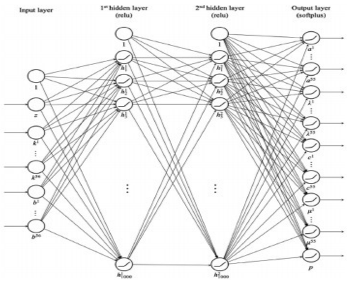
Day 6, Thursday, March 27th, 2025 (Uni Dufour U365)

Time	Main Topics
09:00 - 10:30	Solving PDEs with PINNs (II)
10:30 - 10:45	Coffee Break
10:45 - 12:00	Modeling Sequence Data with RNNs, LSTMs
12:00 - 12:15	Wrap-up of the Course

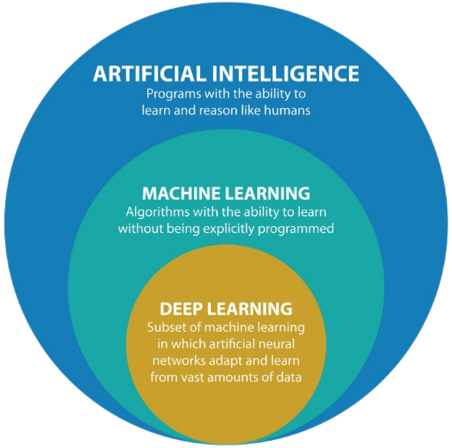
RECAP ON DEEP LEARNING

March 25th, 2023

Simon Scheidegger
simon.scheidegger@unil.ch



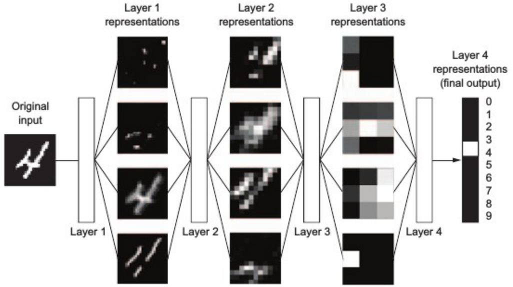
SOME TERMINOLOGY



- **Artificial intelligence (AI)**
  - Can computers be made to “think”—a question whose ramifications we’re still exploring today.
  - A concise definition of the field would be as follows: the effort to automate intellectual tasks normally performed by humans.

- **Machine learning (e.g., supervised ML)**

- **Deep Learning as a particular example of an ML technique**



Figs. from F. Chollet (2018) – ML with Python

TYPES OF MACHINE LEARNING

- **Supervised Learning**
  - assume that training data is available from which they can learn to predict a target feature based on other features (e.g., monthly rent based on area).

    - **Classification**
    - **Regression**

- **Unsupervised Learning**
  - take a given data-set and aim at gaining insights by identifying patterns, e.g., by grouping similar data points.

- **Reinforcement Learning**

BUILDING AN ML ALGORITHM (II)

- **x(i): “input” variables** (living area in this example),

  also called **input features**

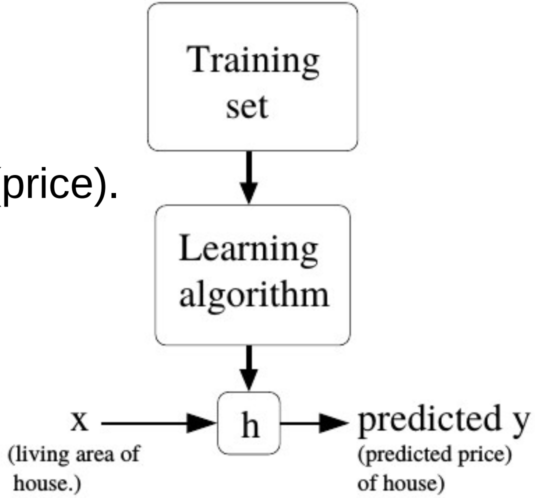
- **y(i): “output”** / **target variable** that we are trying to predict (price).

- **Training example**: a pair (x(i) , y(i)).

- **Training set**: a list of m training examples

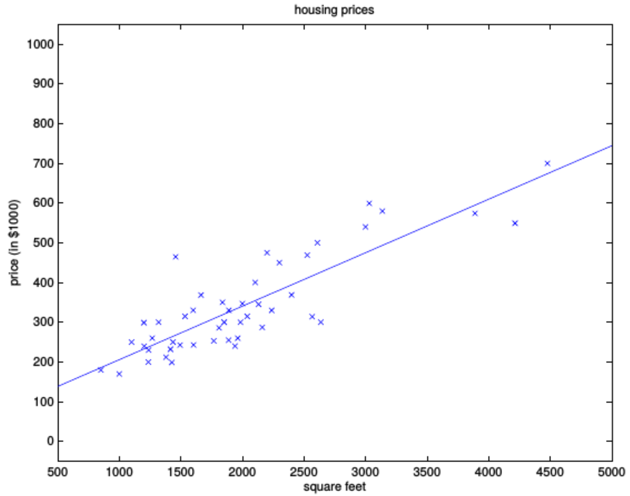
  {(x(i), y (i)); i = 1, . . . , m}

  - To perform supervised learning, we must decide how we’re going to represent **functions/hypotheses *h*** in a computer.



BUILDING AN ML ALGORITHM (III)

- Model / Hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x_1$
  - θ_i's: parameters



- Cost Function:

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$$

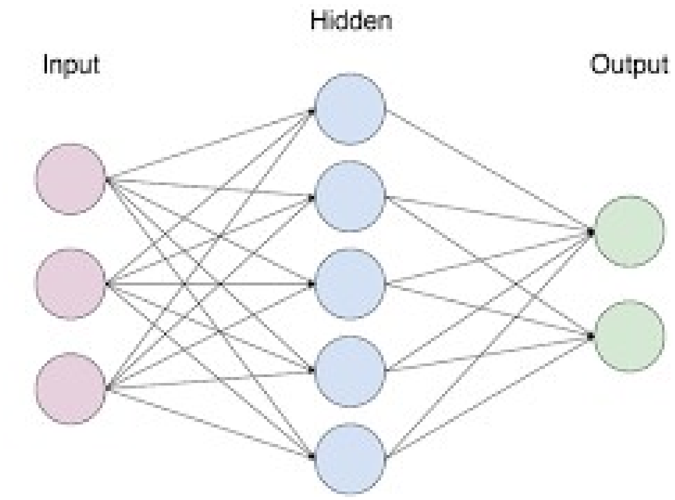
  - Minimize J(θ) in order to obtain the coefficients θ.

BUILDING AN ML ALGORITHM (IV)

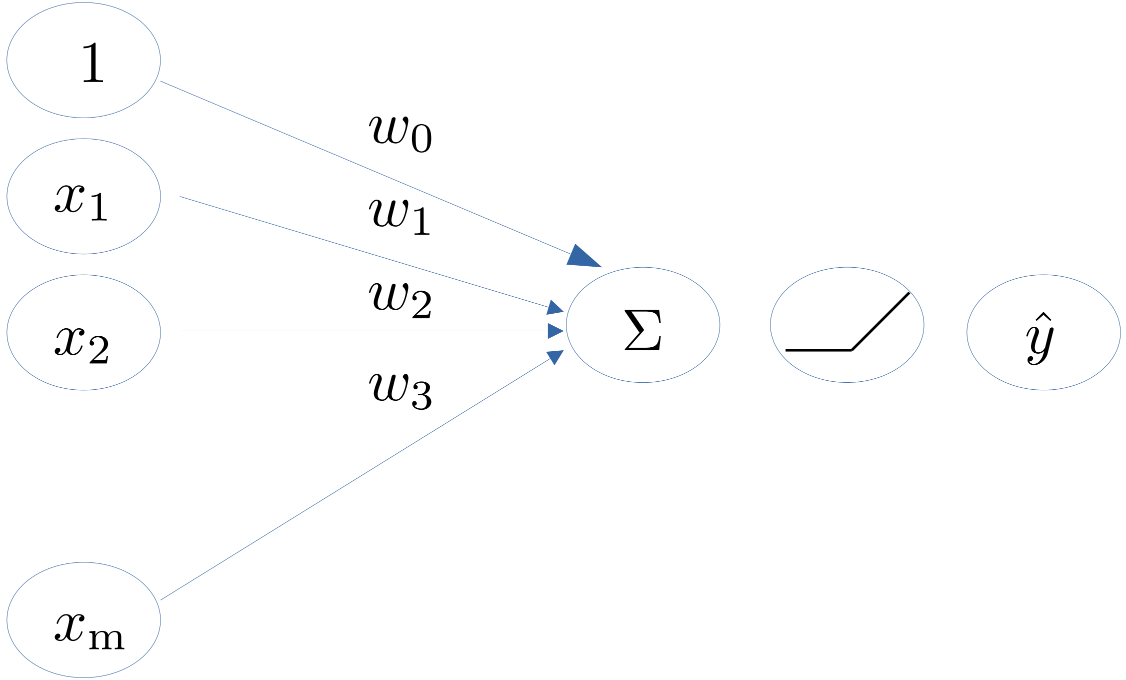
- In General: Machine learning in 3 steps:
  - Choose a **model** $h(x|\theta)$.
  - Define a **cost function** $J(\theta|x)$.
  - **Optimization procedure** to find θ^* that minimizes $J(\theta)$.

- Computationally, we need:
  - data, linear algebra, statistics tools, and optimization routines.

ARTIFICIAL NEURAL NETWORKS



A SINGLE NEURON: THE PERCEPTRON



Inputs Weights Sum Non-Linearity Output

Output Linear combination of inputs

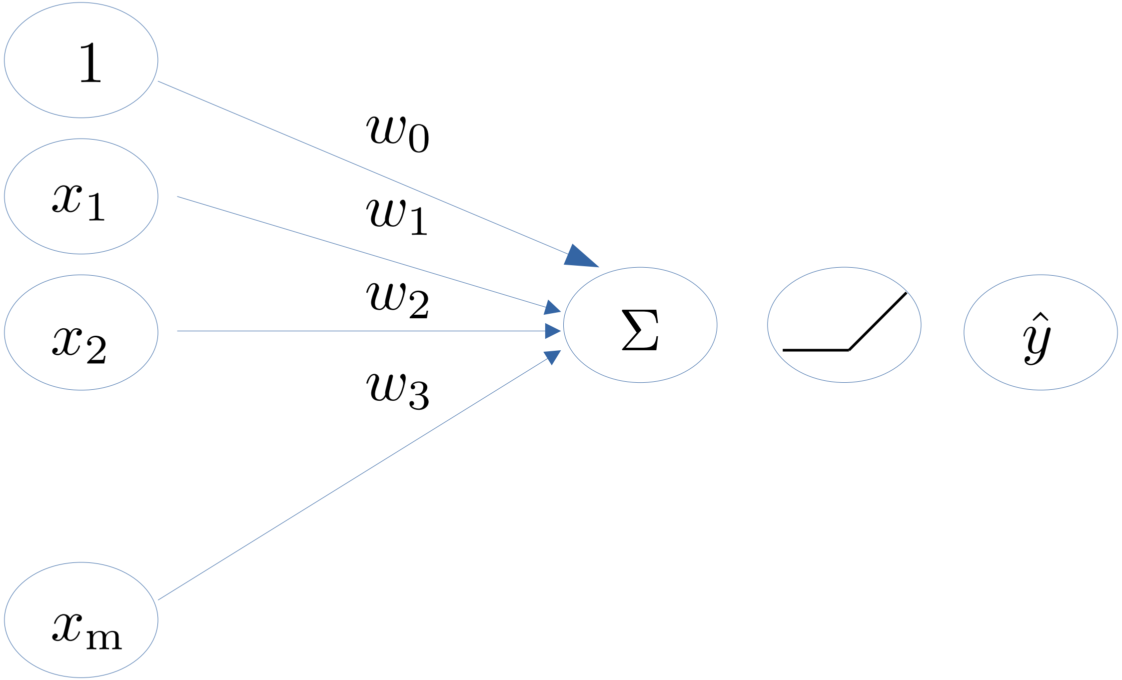
$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i w_i\right)$$

Bias

Non-linear activation function

→ **Bias term allows you to shift your activation function to the left or the right**

THE PERCEPTRON: FORWARD PROPAGATION



$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i w_i\right)$$

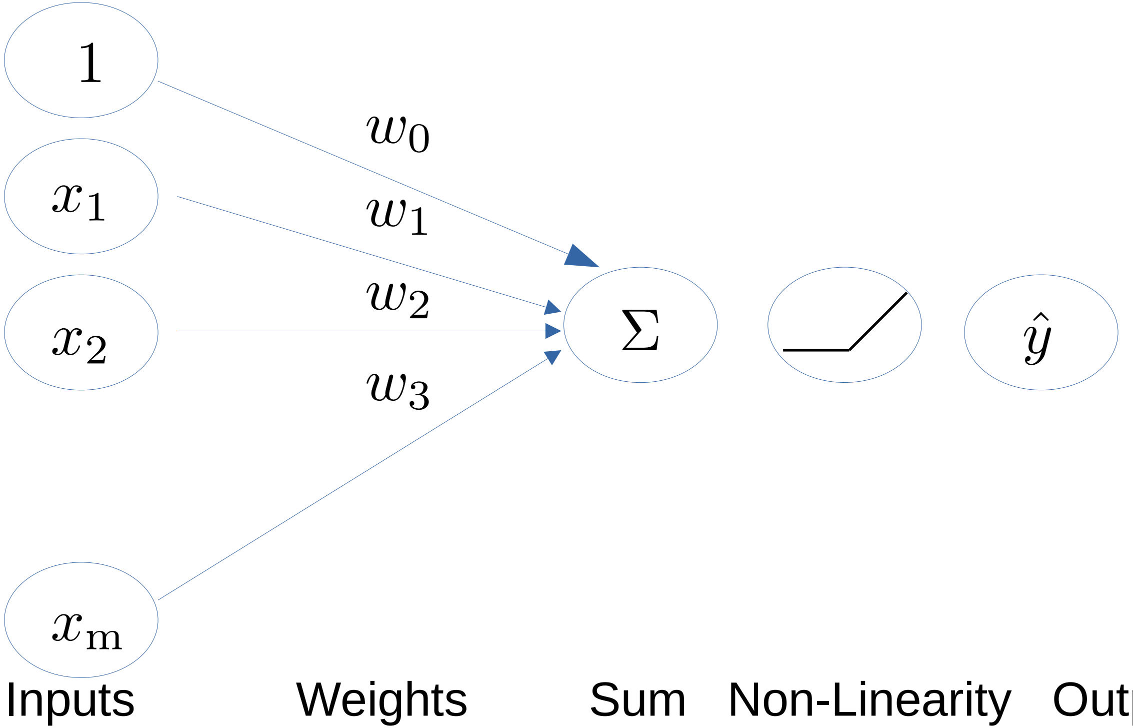
$$\hat{y} = g\left(w_0 + \boldsymbol{X}^T \boldsymbol{W}\right)$$

$$\boldsymbol{X} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix} \text{ and } \boldsymbol{W} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}$$

Inputs Weights Sum Non-Linearity Output

→ **Bias term allows you to shift your activation function to the left or the right**

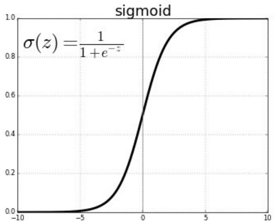
THE PERCEPTRON: FORWARD PROPAGATION



$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i w_i\right)$$

Activation Functions
e.g. sigmoid function

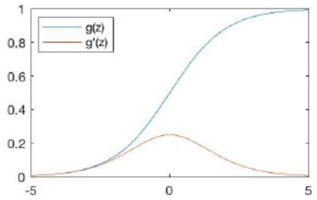
$$g(z) = \sigma(z) = \frac{1}{1+e^{-z}}$$



→ **Bias term allows you to shift your activation function to the left or the right**

FEW ACTIVATION FUNCTIONS

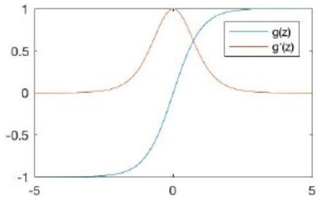
Sigmoid Function



$$g(z) = \frac{1}{1 + e^{-z}}$$

$$g'(z) = g(z)(1 - g(z))$$

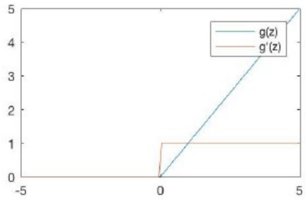
Hyperbolic Tangent



$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

$$g'(z) = 1 - g(z)^2$$

Rectified Linear Unit (ReLU)

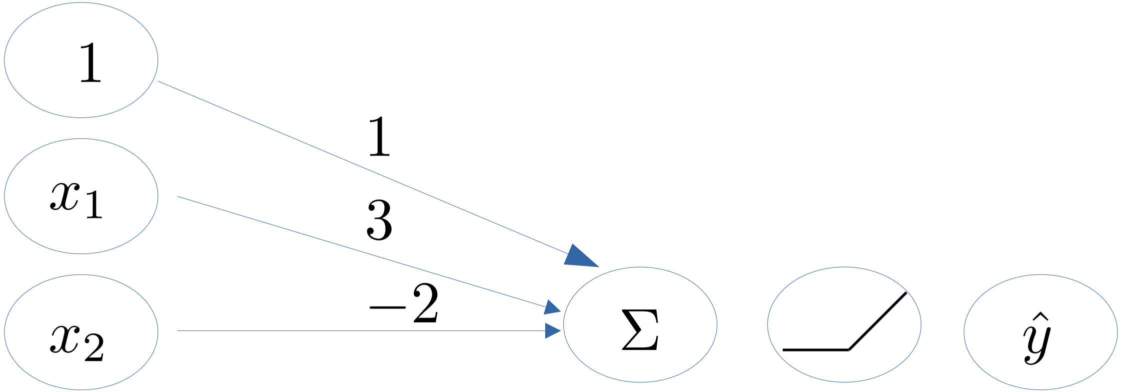


$$g(z) = \max(0, z)$$

$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

- Needs to be differentiable for gradient-based learning (later)
  - Very useful in practice.
  - Sigmoid function, e.g., useful for classification (Probability).

PERCEPTRON – AN EXAMPLE



We have: $w_0 = 1$ and $W = \begin{bmatrix} 3 \\ -2 \end{bmatrix}$

$$\hat{y} = g\left(w_0 + \boldsymbol{X}^T \boldsymbol{W}\right)$$

$$= g\left(1 + \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} 3 \\ -2 \end{bmatrix}\right)$$

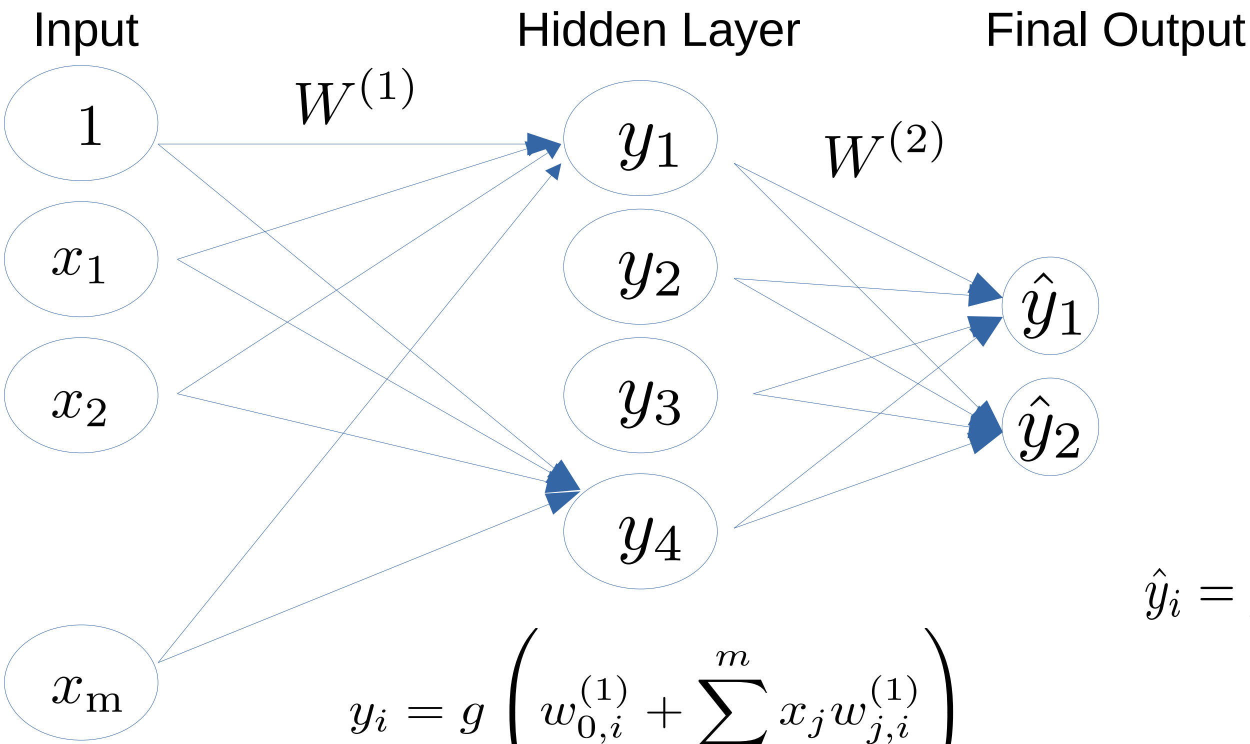
$$\hat{y} = g\left(1 + 3x_1 - 2x_2\right)$$

This is just a line in $2D$

Imagine we have a trained network with weights given.

→ how do we compute the output?

SINGLE HIDDEN LAYER NN



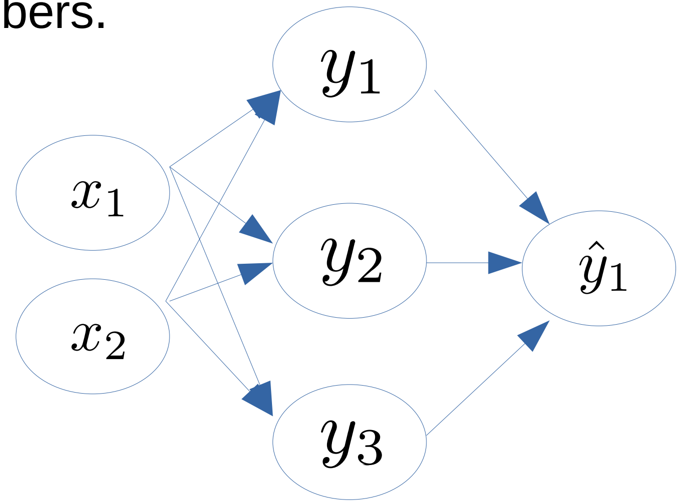
$$\hat{y}_i = g\left(w_{0,i}^{(2)} + \sum_{j=1}^{d} y_j w_{j,i}^{(2)}\right)$$

$$y_i = g\left(w_{0,i}^{(1)} + \sum_{j=1}^{m} x_j w_{j,i}^{(1)}\right)$$

MEAN SQUARED ERROR (MSE)

Mean squared error can be used with regression models that output continuous real numbers.

$$\begin{bmatrix} 80.000, 200.000 \\ 120.000, 400.000 \\ 10.000, 12.000 \\ \dots, \dots \end{bmatrix}$$



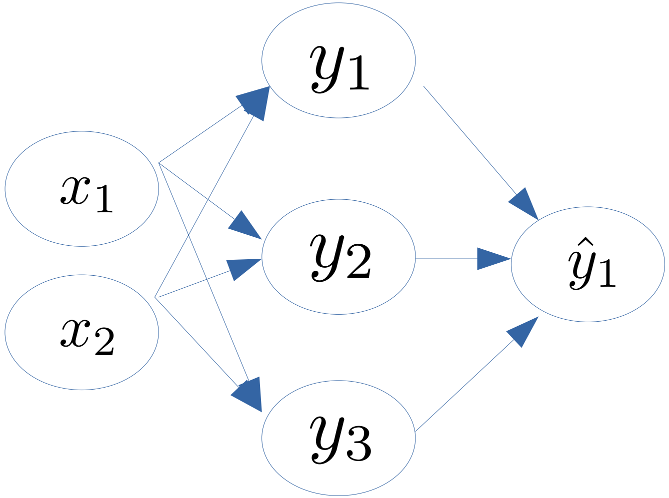
$$\begin{array}{cc} f(x) & y_{true} \\ \begin{bmatrix} 450.000 \\ 250.000 \\ 190.000 \\ \dots, \dots \end{bmatrix} & \begin{bmatrix} 470.000 \\ 220.000 \\ 250.000 \\ \dots, \dots \end{bmatrix} \\ \text{Loan requested} & \text{Loan required} \end{array}$$

$$J(W) = \frac{1}{n} \sum_{i=1}^{n} \Big(\underbrace{y_{true}^{(i)}}_{\text{Actual}} - \underbrace{f\left(x^{(i)}; W\right)}_{\text{Predicted}} \Big)^2$$

BINARY CROSS ENTROPY LOSS

Our example was a classification problem with output (0 or 1)

$$\begin{bmatrix} 80.000, 200.000 \\ 120.000, 400.000 \\ 10.000, 12.000 \\ \ldots, \ldots \end{bmatrix}$$



$$\begin{array}{cc} f(x) & y_{true} \\ \begin{bmatrix} 0.2 \\ 0.9 \\ 0.6 \\ \ldots, \ldots \end{bmatrix} & \begin{bmatrix} 1 \\ 0 \\ 1 \\ \ldots, \ldots \end{bmatrix} \end{array}$$

$$J(W) = \frac{1}{n} \sum_{i=1}^{n} \underbrace{y_{true}^{(i)}}_{\text{Actual}} \log\left(\underbrace{f}_{\text{Predicted}}\left(x^{(i)}; W\right)\right) + \left(1 - \underbrace{y_{true}^{(i)}}_{\text{Actual}}\right) \log\left(1 - \underbrace{f}_{\text{Predicted}}\left(x^{(i)}; W\right)\right)$$

GRADIENT DESCENT IN WEIGHT SPACE

→ We want to find the network weights that achieve the lowest loss!

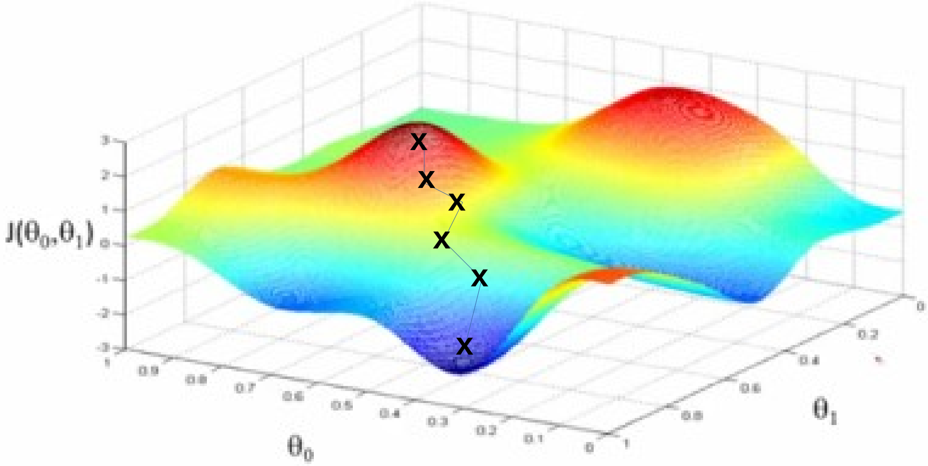
$-W^* = \underset{W}{\operatorname{argmin}} J(W)$

$-$Randomly pick an initial(w_0, w_1)

$-$Compute gradient

$-$Take small steps in the opposite direction of gradient.

$-$Repeat until convergence



GRADIENT DESCENT ALGORITHM

Algorithm

1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(W)}{\partial W}$ ← **Can be computationally expensive**
4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial W}$
5. Return weights

GRADIENT DESCENT ALGORITHM

Algorithm

1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(W)}{\partial W}$
4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial W}$
5. Return weights

All that matters to train a NN. Computationally expensive!

Learning rate

STOCHASTIC GRADIENT DESCENT

Algorithm

I. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$

2. Loop until convergence:

3. Pick single data point i

4. Compute gradient, $\frac{\partial J_i(W)}{\partial W}$

Can be noisy

5. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$

6. Return weights

STOCHASTIC GRADIENT DESCENT

Algorithm

I. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$

2. Loop until convergence:

3. Pick $\boxed{\text{batch of } B \text{ data}}$ points

4. Compute gradient, $\frac{\partial J(W)}{\partial W} = \frac{1}{B} \sum_{k=1}^{B} \frac{\partial J_k(W)}{\partial W}$

5. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$

6. Return weights

COMPUTING GRADIENTS: ERROR BACKPROPAGATION

- How does a small change in one weight (e.g., w_2) affect the final loss J(W)?

$$x \xrightarrow{w_1} z_1 \xrightarrow{w_2} \hat{y} \rightarrow J(W)$$

COMPUTING GRADIENTS: ERROR BACKPROPAGATION

- How does a small change in one weight (e.g., w_2) affect the final loss J(W)?
- Chain rule

$$x \xrightarrow{w_1} z_1 \xrightarrow{w_2} \hat{y} \rightarrow J(W)$$

$$\frac{\partial J(W)}{\partial w_2} = \frac{\partial J(W)}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_2}$$

COMPUTING GRADIENTS: ERROR BACKPROPAGATION

- How does a small change in one weight (e.g., w_2) affect the final loss J(W)?
- Chain rule
- **Repeat this for every weight in the network using gradients from later layers**

$$x \xrightarrow{w_1} z_1 \xrightarrow{w_2} \hat{y} \rightarrow J(W)$$

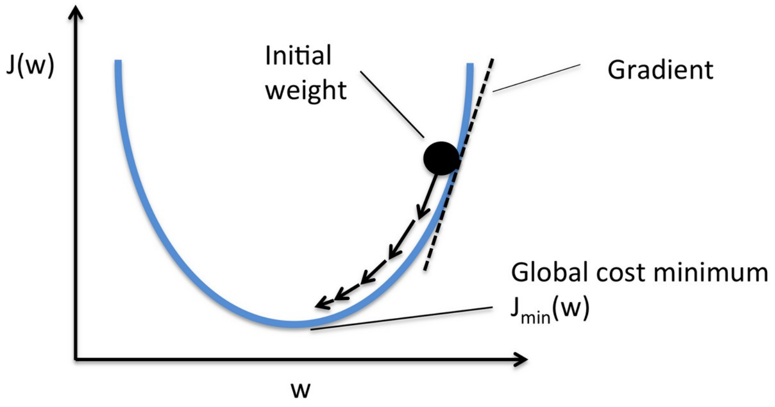
$$\frac{\partial J(W)}{\partial w_1} = \frac{\partial J(W)}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_1} \quad \longrightarrow \quad \frac{\partial J(W)}{\partial w_1} = \frac{\partial J(W)}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z_1} * \frac{\partial z_1}{\partial w_1}$$

LOSS FUNCTION: CAN BE DIFFICULT TO OPTIMIZE

- Remember:
  - Optimization through gradient descent: $$W \leftarrow W - \eta \frac{\partial J(W)}{\partial W}$$
  - How can we set the learning rate?

SETTING THE LEARNING RATE

- Small learning rate converges slowly and gets stuck in false local minima
  - Design an adaptive learning rate that "adapts" to the landscape.



SPLIT THE DATA

- To avoid over-fitting, it is good practice to assess the quality of a model based on test data that **must not be used** for training the model.

- The key idea is to split the available data (randomly) into **training**, **validation**, and **test data**.

SPLITTING THE DATA

- One common approach to reliably assess the quality of a machine learning model and avoid over-fitting is to randomly split the available data into
  - **training data (~70% of the data)**
    is used for determining optimal coefficients.
  - **validation data (~20% of the data) is used for model selection** (e.g., fixing degree of polynomial, selecting a subset of features, etc.)
  - **test data (~10% of the data)** is used to measure the quality that is reported.

A NOTEBOOK

- day4/code/01_recap_week1.ipynb